AI Developer Assignment

JSON Logic Rule Generator (with RAG & Embeddings)

1. Goal

Build an AI-based API that:

1. Takes a **natural-language prompt** and
2. Returns a **JSON Logic rule** using only the provided SAMPLE_STORE_KEYS +
3. Returns a **clear natural-language explanation** of the rule and which fields were used,
4. Uses **embeddings + (optional) RAG** to map user language to the right keys and domain concepts.

No need to implement refinement, validation, or persistence. Just **generate + explain**.

---

2. Inputs: Allowed Keys

The rule generator **must only** use the following keys (exact value strings) in JSON Logic (same as you shared):

```
export const SAMPLE_STORE_KEYS = [
  { value: 'business.address.pincode', label: 'Business Pincode', group: 'business' },
  { value: 'business.address.state', label: 'Business State', group: 'business' },
  { value: 'business.vintage_in_years', label: 'Business Vintage In Years', group: 'business' },
  { value: 'business.commercial_cibil_score', label: 'Commercial Cibil Score', group:
'business' },
  { value: 'primary_applicant.age', label: 'Primary Applicant Age', group: 'primary_applicant'
},
  { value: 'primary_applicant.monthly_income', label: 'Primary Applicant Monthly Income',
group: 'primary_applicant' },
  { value: 'primary_applicant.tags', label: 'Primary Applicant Tags', group: 'primary_applicant'
},
  { value: 'bureau.score', label: 'Bureau Score', group: 'bureau' },
  { value: 'bureau.is_ntc', label: 'Is New to Credit?', group: 'bureau' },
  { value: 'bureau.overdue_amount', label: 'Overdue Amount', group: 'bureau' },
  { value: 'bureau.dpd', label: 'DPD', group: 'bureau' },
  { value: 'bureau.active_accounts', label: 'Active Accounts', group: 'bureau' },
  { value: 'bureau.enquiries', label: 'Enquiries', group: 'bureau' },
  { value: 'bureau.suit_filed', label: 'Suit Filed', group: 'bureau' },
  { value: 'bureau.wilful_default', label: 'Wilful Default', group: 'bureau' },
  { value: 'banking.abb', label: 'ABB', group: 'banking' },
```

```
  { value: 'banking.avg_monthly_turnover', label: 'Avg Monthly Turnover', group: 'banking' },
  { value: 'banking.total_credits', label: 'Total Credits', group: 'banking' },
  { value: 'banking.total_debits', label: 'Total Debits', group: 'banking' },
  { value: 'banking.inward_bounces', label: 'Inward Bounces', group: 'banking' },
  { value: 'banking.outward_bounces', label: 'Outward Bounces', group: 'banking' },
  { value: 'gst.registration_age_months', label: 'Registration Age Months', group: 'gst' },
  { value: 'gst.place_of_supply_count', label: 'Place Of Supply Count', group: 'gst' },
  { value: 'gst.is_gstin', label: 'Is GSTIN', group: 'gst' },
  { value: 'gst.filing_amount', label: 'Filing Amount', group: 'gst' },
  { value: 'gst.missed_returns', label: 'Missed Returns', group: 'gst' },
  { value: 'gst.monthly_turnover_avg', label: 'Monthly Turnover Avg', group: 'gst' },
  { value: 'gst.turnover', label: 'Turnover', group: 'gst' },
  { value: 'gst.turnover_growth_rate', label: 'Turnover Growth Rate', group: 'gst' },
  { value: 'gst.output_tax_liability', label: 'Output Tax Liability', group: 'gst' },
  { value: 'gst.tax_paid_cash_vs_credit_ratio', label: 'Tax Paid Cash Vs Credit Ratio', group:
'gst' },
  { value: 'gst.high_risk_suppliers_count', label: 'High Risk Suppliers Count', group: 'gst' },
  { value: 'gst.supplier_concentration_ratio', label: 'Supplier Concentration Ratio', group: 'gst'
},
  { value: 'gst.customer_concentration_ratio', label: 'Customer Concentration Ratio', group:
'gst' },
  { value: 'itr.years_filed', label: 'Years Filed', group: 'itr' },
  { value: 'foir', label: 'FOIR', group: 'metrics' },
  { value: 'debt_to_income', label: 'Debt To Income', group: 'metrics' },
];
```

---

3. Scope of Work

3.1 API endpoints (minimal)

Implement **one** of the following designs (your choice):

Option A: Single endpoint

POST /generate-rule

Request:

```
{
  "prompt": "Approve if bureau score > 700 and business vintage at least 3 years and
applicant age between 25 and 60.",
  "context_docs": ["optional domain policy text here"]
}
```

Response:

```
{
  "json_logic": { /* valid JSON Logic rule */ },
  "explanation": "Short natural-language explanation of the rule.",
  "used_keys": ["bureau.score", "business.vintage_in_years", "primary_applicant.age"],
  "key_mappings": [
    { "user_phrase": "bureau score", "mapped_to": "bureau.score", "similarity": 0.93 },
    { "user_phrase": "business vintage", "mapped_to": "business.vintage_in_years",
"similarity": 0.91 }
  ],
  "confidence_score": 0.0
}
```

> confidence_score can be simple (0–1): based on mapping similarity, LLM confidence, etc.

3.2 Functional requirements

1. **Generate valid JSON Logic**
    - Use a JSON Logic structure (e.g., {"and": [...]},"if", "or", ">", "<=", "in", "==" etc.).
    - Only use keys from SAMPLE_STORE_KEYS under "var".
2. **Explain the rule**
    - Explain in 1–3 sentences what the rule does in plain English.
    - Mention thresholds and important conditions.
3. **Key mapping using embeddings**
    - Build embeddings for each SAMPLE_STORE_KEYS item using {label, value, group} text.
    - Given the user prompt, use embeddings to map phrases to keys.
    - Example: “credit score” → bureau.score; “business age” → business.vintage_in_years.
    - Return mapping plus similarity in key_mappings.
4. **Optional but preferred: simple RAG**
    - Assume there are 2–3 short “policy documents” (can be markdown strings in code).
    - Use embeddings + a simple vector index (FAISS / in-memory cosine) to retrieve the top relevant policy snippets.
    - Feed these into the LLM prompt to help guide the rule.
    - Example: A policy doc says “Minimum bureau score must be 600” and prompt says “high credit score” → you prefer 600+ thresholds.
5. **Unknown or unmappable fields**
    - If the user mentions something not in SAMPLE_STORE_KEYS (e.g., “loan amount”):
        - Either map to the closest key **only if** similarity is above a threshold (e.g., 0.75),

- Or return an error explaining that field is not available and list top 3 suggestions with similarity scores.

---

4. Example Prompts & Expected JSON Logic

Example 1

Prompt:

> Approve if bureau score > 700 and business vintage at least 3 years and applicant age between 25 and 60.

Expected JSON Logic (one valid version):

```
{
  "and": [
    { ">": [ { "var": "bureau.score" }, 700 ] },
    { ">=": [ { "var": "business.vintage_in_years" }, 3 ] },
    {
      "and": [
        { ">=": [ { "var": "primary_applicant.age" }, 25 ] },
        { "<=": [ { "var": "primary_applicant.age" }, 60 ] }
      ]
    }
  ]
}
```

Example 2

Prompt:

> Flag as high risk if wilful default is true OR overdue amount > 50000 OR bureau.dpd >= 90.

Expected JSON Logic:

```
{
  "or": [
    { "==": [ { "var": "bureau.wilful_default" }, true ] },
    { ">": [ { "var": "bureau.overdue_amount" }, 50000 ] },
    { ">=": [ { "var": "bureau.dpd" }, 90 ] }
  ]
}
```

Example 3

Prompt:

> Prefer applicants with tag ‘veteran’ OR with monthly_income > 1,00,000.

Expected JSON Logic:

```
{
  "or": [
    { "in": [ "veteran", { "var": "primary_applicant.tags" } ] },
    { ">": [ { "var": "primary_applicant.monthly_income" }, 100000 ] }
  ]
}
```

Your API **does not need to run these rules**, but it must be syntactically valid JSON Logic and must correctly reference the allowed keys.

---

5. RAG & Embeddings — What We Expect

Minimum expectations:

- Use an embedding model (can be OpenAI or open-source) to:
  - Embed SAMPLE_STORE_KEYS.
  - Embed the user prompt.
- Use cosine similarity (or similar) to:
  - Find which keys are most relevant to the prompt.
  - Provide key_mappings and similarity values in the response.
- Optional: Add a very lightweight RAG layer:
  - Hardcode a small list of “policy docs” into the repo.
  - Embed + store them; on each request, retrieve the most relevant doc text and include it in the LLM prompt.

---

6. Deliverables

- Source code (any backend stack; Node/Express, Python/FastAPI, etc.).
- README with:
  - How to run locally.
  - Example curl/postman request to /generate-rule.
- A few **hard-coded test examples** (you can put them in a script or tests folder) using the prompts above and printing:
  - Prompt
  - JSON Logic
  - Explanation
  - Key mappings

---

7. Evaluation Criteria (Simplified)

Out of 100:

- **Correct JSON Logic & key usage (35 pts)**
  - Valid JSON, valid JSON Logic structure, only allowed keys.
- **Explanations (15 pts)**
  - Clear, accurate, concise rule explanation.
- **Embeddings & key mapping (30 pts)**
  - Correct usage of embeddings to pick keys.
  - Crisp key_mappings output with similarity metrics.
- **Code quality & structure (20 pts)**
  - Clean, readable code, sensible separation of concerns.
  - README and example requests.